SMT-based Bounded Model Checking for Real-time Systems *

Liang Xu[1,2]
[1]State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]Graduate University of Chinese Academy of Sciences, Beijing, China
xul@ios.ac.cn

Abstract

SAT-based bounded model checking has a high complexity in dealing with real-time systems. SMT solvers can generalize SAT solving by adding the ability to handle arithmetic and other decidable theories. With this advantage, if we use SMT in bounded model checking for real-time systems instead of SAT, the clocks can be represented as integer or real variables directly and clock constraints can be represented as linear arithmetic expressions. This makes the checking procedure more efficient. We use TCTL to specify the properties of real-time systems.

1 Introduction

In symbolic model checking, Binary Decision Diagrams (BDD's) [6] are used to represent system variables. Lots of system behaviors can be explored by one BDD computation. The technique is used in many model checking tools, such as NuSMV [7]. But the size of BDD may grow significantly as the number of variables increased. Checking a system with a large number of variables remains a difficult problem for BDD-based model checking tools. On the other hand, Bounded Model Checking (BMC) based on Boolean Satisfiability (SAT) has been introduced as a complementary technique to BDD's model checking for combating the state explosion problem. The basic idea of BMC presented in [4, 5, 19] is to restrict the general model checking problem to a bounded one. Instead of finding out whether the system M violates the property ψ, we only need to know whether the system M has counterexamples. The method's efficiency is based upon the fact that if there is a counterexample, then we may find it in a small portion of its state space [4, 5].

*Supported by the National Natural Science Foundation of China under Grant No. 60573012 and 60721061, and the National Grand Fundamental Research 973 Program of China under Grant No. 2002cb312200.

The verification of real-time systems is a very important and challenging problem. Since the interest in automated verification has been moving towards to real-time systems, many researches were done in recent years. [20, 23] transform the Timed Automaton (TA) to a Region Graph (RG) [3], which is based on dense-time approach [15, 17, 8], then encode a Timed CTL (TCTL) [3] formula to a propositional formula and use SAT solvers to check it. TSMV [18] uses Timed Kripke Structure (TKS) as its model, which is based on discrete-time approach [2, 14, 11], and solves problems on BDD-based methods. Either SAT-based or BDD-based method for real-time systems, needs to encode clocks and clock constraints into boolean formulae. After this encoding, the clocks' characteristics disappeared which means the whole checking process has no time information to use. In order to overcome this disadvantage, we use Satisfiability Modula Theories (SMT) instead of SAT to do the check.

The SMT problem is a generalization of the SAT problem where boolean variables are replaced by predicates from various background theories, such as linear real and integer arithmetic. So, we can use real or integer variables to represent clocks and linear arithmetic expressions to represent clock constraints instead of boolean formulae, which preserve the time characteristics in the checking process. There are some related works for SMT-based BMC, such as [1] which is dealing with programming languages. Our contribution is that we combine the advantages of BMC and SMT to verify real-time systems. By doing so, the encoding of clock variables and clock constraints has less effect on BMC's efficiency for real-time systems.

The rest of this paper is organized as follows. The notions of real-time systems and TECTL [3] are introduced in the next Section . The SMT-based BMC approach is given in Section 3 and the experimental results are summarized in Section 4. Concluding remarks are given in Section 5.

1550-6002/08 $25.00 © 2008 IEEE
DOI 10.1109/QSIC.2008.10

2 Preliminaries

2.1 Real-time Systems

Real-time systems not only contain discrete variables but also have dense-time clocks which have a real domain and continuously increase at a uniform rate. Clocks are usually set to zero at the beginning and can be reset at any time. Real-time systems can often be modeled as a TA or simply modeled as a TKS.

2.1.1 Timed Automata

TA is a finite-state machine equipped with a set of clocks. Hereafter, the set $\mathcal{AP} = \{p_1, p_2, \ldots\}$ denotes *atomic propositions*, $\mathbb{N}$ and $\mathbb{R}^+$ denote the set of natural numbers and non-negative real numbers respectively.

Let $\mathcal{X}$ be a finite set of variables called clocks. A clock valuation is a function $v : \mathcal{X} \to \mathbb{R}^+$, which assigns a non-negative real number $v(x)$ to each clock $x \in \mathcal{X}$. For a subset $\mathcal{Y}$ of $\mathcal{X}$ by $v[\mathcal{Y} := 0]$ we mean the valuation v' such that $\forall x \in \mathcal{Y}, v'(x) = 0$ and $\forall x \in \mathcal{X} \setminus \mathcal{Y}, v'(x) = v(x)$. For $\delta \in \mathbb{R}^+$, $v + \delta$ denotes the valuation v'' such that $\forall x \in \mathcal{X}$, $v''(x) = v(x) + \delta$. The set $\Psi_{\mathcal{X}}$ of clock constraints over the set of clocks $\mathcal{X}$ is defined as follows:

$$\psi ::= x \prec c \,|\, x - x' \prec c \,|\, \psi \wedge \psi' \,|\, \neg\psi$$

where $x, x' \in \mathcal{X}$, $\prec \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

A clock valuation $v \in \mathcal{V}_{\mathcal{X}}$, $\mathcal{V}_{\mathcal{X}}$ denotes the set of all valuations, satisfies the clock constraint $\psi \in \Psi_{\mathcal{X}}$ denotes:

$v \models x \prec c$	iff	$v(x) \prec c$
$v \models x - x' \prec c$	iff	$v(x) - v'(x) \prec c$
$v \models \psi \wedge \psi'$	iff	$v \models \psi$ and $v \models \psi'$
$v \models \neg\psi$	iff	$v \not\models \psi$

$[[\psi]] = \{v \in \mathcal{V}_{\mathcal{X}} \,|\, v \models \psi\}$ is the set of valuations that satisfy ψ.

Definition 1 *A TA is a tuple $\langle \mathcal{S}, \mathcal{X}, \Sigma, s_0, \varepsilon, I\rangle$ where: $\mathcal{S}$ is a finite set of locations; $\mathcal{X}$ is a finite set of clocks; Σ is a finite set of labels; $s_0 \in \mathcal{S}$ is an initial location; ε is a finite set of transition relations, $\varepsilon \subseteq \mathcal{S} \times \Sigma \times \Psi_{\mathcal{X}} \times 2^{\mathcal{X}} \times \mathcal{S}$; $I : \mathcal{S} \to \Psi_{\mathcal{X}}$ is a state invariant function.*

Each element $e \in \varepsilon$ is denoted by $e := s \xrightarrow{l,\psi,\mathcal{Y}} s'$. This represents a transition from location s to location s' on the input label $l \in \Sigma$. The set $\mathcal{Y} \subseteq \mathcal{X}$ gives the clocks to be reset with this transition. $\psi \in \Psi_{\mathcal{X}}$ is the enabling condition for e.

Let c_{max} be the largest constant appearing in $\Psi_{\mathcal{X}}$. For $x \in \mathcal{X}$, $frac(v(x))$ denotes the fractional part of $v(x)$, and $\lfloor v(x) \rfloor$ denotes its integral part.

Definition 2 *For two clock valuations v and v', $v \simeq_{\Psi_{\mathcal{X}}} v'$ iff for all $x, y \in \mathcal{X}$ follows the conditions below:*

- $v(x) > c_{max} \to v'(x) > c_{max}$
- $v(x) \leq c_{max} \Rightarrow ((\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor) \wedge (frac(v(x)) = 0 \to frac(v'(x)) = 0))$
- $\forall (x - y \prec c) \in \Psi_{\mathcal{X}}.((c \leq c_{max}) \Rightarrow (v \models x - y \prec c \to v' \models x - y \prec c))$

We use $[v]$ to denote the equivalence class of the relation $\simeq_{\Psi_{\mathcal{X}}}$ to which v belongs. Such a class is called a zone. The set of all the zones is denoted by $\mathcal{Z}(n)$. A zone $[v]$ is final iff $v(x) > c_{max}$ for all $v \in [v]$ and $x \in \mathcal{X}$. A zone $[v]$ satisfies the clock constraint $\psi \in \Psi_{\mathcal{X}}$, if $[v] \models \psi$ iff $\forall v' \in [v], v' \models \psi$.

Definition 3 *The RG of a TA is a finite structure $\langle \mathcal{Q}, q^0, \to, \mathcal{L}\rangle$:*

- *$\mathcal{Q} = \{(s, [v]) | (s, [v]) \in \mathcal{S} \times \mathcal{Z}(n)\}$, is the set of states.*
- *$q^0 \in \mathcal{Q}$ is the initial state.*
- *$\to$ is defined as follows:*
  - *$(s, [v]) \xrightarrow{l} (s', [v'])$ iff $\forall e : s \xrightarrow{l,\psi,\mathcal{Y}} s' \in \varepsilon$ such that $s = source(e)$, $s' = target(e)$, $[v] \models \psi$, $[v'] = [v[\mathcal{Y} := 0]]$ and $v' \models I(s')$.*
  - *$(s, [v]) \xrightarrow{\delta} (s, [v'])$ iff $[v'] \models I(s)$ and $[v'] = [v] + \delta$ or $[v'] = [v]$ if $[v]$ is final.*
- *$\mathcal{L} : \mathcal{Q} \to 2^{\mathcal{AP}}$, is a labeling function that maps each state of $\mathcal{Q}$ to a set of atomic propositions true at that state.*

A state q is deadlock if there is no delay $\delta \in \mathbb{R}^+$ and an action $l \in \Sigma$ such that $q \xrightarrow{\delta} q' \xrightarrow{l} q''$, for some $q', q'' \in \mathcal{Q}$.

2.1.2 Timed Kripke Structure

TKS is an extension of Kripke Structure (KS). Each of its transition is labeled by a non-negative integer.

Definition 4 *A TKS is a tuple $\langle \mathcal{S}, s_0, \mathcal{R}, \mathcal{L}\rangle$ where: $\mathcal{S}$ is a finite set of states; $s_0 \in \mathcal{S}$ is an initial state; $\mathcal{R} \subseteq \mathcal{S} \times \mathbb{N} \times \mathcal{S}$ is a finite set of transitions labeled by a natural number, called the duration of the transition; $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{AP}}$ which is similar to the $\mathcal{L}$ in Definition 2.*

A path π in TKS is a infinite sequence $s_0 \xrightarrow{d_1} s_1 \xrightarrow{d_2} s_2 \xrightarrow{d_3} \cdots$, where $s_0, s_1, s_2, \cdots \in \mathcal{S}$. For such a path, and for $n \in \mathbb{N}$, let $\pi(n)$ denotes the nth state s_n, and π^n denotes the nth suffix $s_n \xrightarrow{d_{n+1}} s_{n+1} \xrightarrow{d_{n+2}} \cdots$. Finally, for $n \leq m \in \mathbb{N}$, let $\pi[n..m]$ denotes the finite sequence $s_n \xrightarrow{d_{n+1}} s_{n+1} \xrightarrow{d_{n+2}} \cdots s_m$ with $m - n$ transitions and $m - n + 1$ states. The duration $D_\pi[n..m]$ of such a finite sequence is $d_n + d_{n+1} + \cdots + d_{m-1}$ ($D_\pi = 0$ when $n = m$). In TKS, $D_\pi[0..n]$ represents the time elapsed from s_0 to s_n and there is no reset operator which is different from TA.

2.2 TECTL

Computation tree logic (CTL) is a propositional branching-time temporal logic introduced by [10] as a specification language for finite state systems. TECTL is ECTL (the restriction of CTL to existential path quantifiers) with timing constraints. TECTL formulae are defined as follows:

$$\alpha, \beta := p \,|\, \neg\alpha \,|\, \alpha \vee \beta \,|\, \alpha \wedge \beta \,|\, EF_{\varphi}\alpha \,|\, EG_{\varphi}\alpha \,|\, E(\alpha U_{\varphi}\beta)$$

where $p \in \mathcal{AP}$ and φ called the timing constraint with the form of "$\leq n$", "$= n$", "$\geq n$" for n a value in the relevant domain (here $\mathbb{N}$). Let I^i_{φ} be the predicate that state i satisfies the timing constraint φ where $i \in \mathbb{N}$.

Definition 5 (Semantics) *Let $s_i \in \mathcal{S}$, $p \in \mathcal{AP}$ and ψ, α, β TECTL formulae. $s_i \models \psi$ denotes that ψ is true at the state s_i. The relation $\models$ is defined as follows:*

$s_i \models p$	*iff*	$p \in \mathcal{L}(s_i)$
$s_i \models \neg\alpha$	*iff*	$s \not\models \alpha$
$s_i \models \alpha \wedge \beta$	*iff*	$(s_i \models \alpha) \wedge (s_i \models \beta)$
$s_i \models \alpha \vee \beta$	*iff*	$(s_i \models \alpha) \vee (s_i \models \beta)$
$s_i \models EF_{\varphi}\alpha$	*iff*	$\exists\pi.(s_0 = s_i \wedge \exists k \geq 0.(I^k_{\varphi} \wedge s_k \models \alpha))$
$s_i \models EG_{\varphi}\alpha$	*iff*	$\exists\pi.(s_0 = s_i \wedge \forall k \geq 0.(I^k_{\varphi} \rightarrow s_k \models \alpha))$
$s_i \models E(\alpha U_{\varphi}\beta)$	*iff*	$\exists\pi.(s_0 = s_i \wedge \exists k \geq 0.((I^k_{\varphi} \wedge s_k \models \beta) \wedge 0 \leq \forall j < k.(s_j \models \alpha)))$

As an example, $s_i \models EF_{\varphi}\alpha$ means that there is a path π with its first state equal to s_i and can reach a state satisfying α and the timing constraint φ.

3 SMT-based Bounded Model Checking

3.1 Bounded Semantics of TECTL

In order to define the bounded semantics of TECTL we give some notions first. Let $M = \langle S, s_0, T, L\rangle$ be a model has different semantics in different models. If TA is used as the model, we have to transform the TA to a corresponding RG, then the elements in M stands for the corresponding ones in Definition 3 and $dead(w_k)$ means the state w_k is a deadlock. If the model is represented by TKS, the elements in M stands for the corresponding ones in Definition 4, $dead(w_k)$ is set $true$ if there is a $k \in \mathbb{N}^+$, where $\mathbb{N}^+ = \{1, 2, \ldots\}$, $D_{\pi}[0..k] > n$ and the timing constraint is in the form of "$\leq n$" or "$= n$", else $dead(w_k)$ is set $false$. A k-path $\pi = w_0 \cdots w_k$ of M is a finite sequence of states such that $(w_i, w_{i+1}) \in T$ for $i = 0, \ldots, k-1$ and w_i denotes the ith state in path π. The k-model for M is a structure $M_k = \langle S, s_0, Path_k, L\rangle$, where $Path_k$ is the set of all different k-paths of M. We use $loop(\pi)$ to denote $\{l | l \leq k \wedge (w_k, w_l) \in T\}$.

Definition 6 (Bounded Semantics) *Let M_k be a k-model of M, s a state, $p \in \mathcal{AP}$ and ψ, α, β TECTL formulae. $M_k, s \models_k \psi$ denotes that ψ is true at the state s in M_k. The relation $\models_k$ is defined as follows:*

$M_k, s \models_k p$	*iff*	$p \in L(s)$
$M_k, s \models_k \neg p$	*iff*	$p \notin L(s)$
$M_k, s \models_k \alpha \wedge \beta$	*iff*	$(M_k, s \models_k \alpha) \wedge (M_k, s \models_k \beta)$
$M_k, s \models_k \alpha \vee \beta$	*iff*	$(M_k, s \models_k \alpha) \vee (M_k, s \models_k \beta)$
$M_k, s \models_k EF_{\varphi}\alpha$	*iff*	$\exists\pi.(w_0 = s \wedge 0 \leq \exists i \leq k.(I^i_{\varphi} \wedge M_k, w_i \models_k \alpha))$
$M_k, s \models_k EG_{\varphi}\alpha$	*iff*	$\exists\pi.(w_0 = s \wedge (loop(\pi) \neq \emptyset \vee dead(w_k)) \wedge 0 \leq \forall i \leq k'.(I^i_{\varphi} \rightarrow M_k, w_i \models_k \alpha))$
$M_k, s \models_k E(\alpha U_{\varphi}\beta)$	*iff*	$\exists\pi.(w_0 = s \wedge 0 \leq \exists i \leq k.((I^i_{\varphi} \wedge M_k, w_i \models_k \beta) \wedge 0 \leq \forall j < i.(M_k, w_j \models_k \alpha)))$

In TA model, $k' = k$. In TKS model, $k' = k$ if the timing constraint is in the form of "$\leq n$" or "$= n$", else $k' = 2k - l'$ where $l' = min(loop(\pi))$.

ψ is true in k-model M_k, denoted $M_k \models_k \psi$, iff ψ is true at the initial state of the model M_k. Similar to [19], we have the following two theorems.

Theorem 1 *Let $M_k = \langle S, s_0, Path_k, L\rangle$ be a k-model and ψ a TECTL formula. If $M_k \models_k \psi$, then $M_{k+1} \models_{k+1} \psi$.*

Theorem 2 *Let $M = \langle S, s_0, T, L\rangle$ be a model, ψ a TECTL formula, $k = |M|$ and M_k the k-model of M. Then $M \models \psi$ iff $M_k \models_k \psi$.*

With Theorem 2, we are able to reduce the model checking problem $M \models \psi$ to BMC problem $M_k \models_k \psi$.

3.2 SMT-based Bounded Model Checking for TECTL

In this subsection, we show how to encode k-model and TECTL formulae.

Given an TECTL formula ψ, a model M and a bound k, SMT-based BMC approach for TECTL needs to generate and solve a semi-propositional formula (as the time variables and clock constraints are not translated into boolean formulae) $[[M, \psi]]_k := [[M^{\psi, s_0}]]_k \wedge [[\psi]]_k$ where $[[M^{\psi, s_0}]]_k$ represents the transition relations of the k-paths in M_k and $[[\psi]]_k$ specifies which k-paths satisfies ψ. The satisfiability of the semi-propositional formula on some k implies that M satisfies ψ. Otherwise, ψ do not hold in M.

In order to construct $[[M, \psi]]_k$, we first give some useful notions. Let w be a vector state variable, $w = (w[1], \ldots, w[n])$, where $w[i]$ for $i = 1, \ldots, n$ are propositional variables and n depends on the size of the model, $n = \lceil log_2(|M|)\rceil$. A state can be represented by a truth assignment to $(w[1], \ldots, w[n])$. When we talk about a state w, we mean the state represented by w with a given assignment. The equality $w_i \equiv w_j$ is defined by $\bigwedge_{m=1}^{n} w_i[m] \Leftrightarrow w_j[m]$. Let $k \geq 0$, $w_{j,i}$ represents the jth state on the ith path. So we use $w_{0,i}, \ldots, w_{k,i}$ to represent the $k+1$ states of

the ith k-path for each $i \in \{1, \ldots, N_k(\psi)\}$, where $N_k(\psi)$ is the number of different k-paths needed for checking the formula ψ, and it is defined as follows:

- $N_k(p) = N_k(\neg p) = 0$, where $p \in \mathcal{AP}$
- $N_k(\alpha \vee \beta) = max\{N_k(\alpha), N_k(\beta)\}$
- $N_k(\alpha \wedge \beta) = N_k(\alpha) + N_k(\beta)$
- $N_k(EF_\varphi \alpha) = N_k(\alpha) + 1$
- $N_k(EG_\varphi \alpha) = (k+1) \cdot N_k(\alpha) + 1$
- $N_k(E(\alpha U_\varphi \beta)) = k \cdot N_k(\alpha) + N_k(\beta) + 1$

Definition 7 (Transformation of the k-model) *Let $M_k = \langle S, Path_k, s, L\rangle$ be the k-model of M and ψ an TECTL formula. $[[M^{\psi,s}]]_k$ is defined as follows:*

$$[[M^{\psi,s}]]_k := I(s) \wedge \bigwedge_{i=1}^{N_k(\psi)} \bigwedge_{j=0}^{k-1} T(w_{j,i}, w_{j+1,i})$$

where $I(s)$ is true when s is the initial state.

Definition 8 (Transformation of TECTL formulae) *Let $p \in \mathcal{AP}$, ψ, α, β TECTL formulae, m and n represent the state's number and path's number respectively. $[[\psi]]_k^{[m,n]}$ is defined as follows:*

$$\begin{array}{lcl}
[[p]]_k^{[m,n]} & := & p(w_{m,n}), \\
[[\neg p]]_k^{[m,n]} & := & \neg p(w_{m,n}), \\
[[\alpha \wedge \beta]]_k^{[m,n]} & := & [[\alpha]]_k^{[m,n]} \wedge [[\beta]]_k^{[m,n]}, \\
[[\alpha \vee \beta]]_k^{[m,n]} & := & [[\alpha]]_k^{[m,n]} \vee [[\beta]]_k^{[m,n]}, \\
[[EF_\varphi \alpha]]_k^{[m,n]} & := & \bigvee_{i=1}^{N_k(\varphi)}((w_{m,n} \equiv w_{0,i}) \wedge \\
 & & \bigvee_{j=0}^{k}(I_\varphi^j \wedge [[\alpha]]_k^{[j,i]})), \\
[[EG_\varphi \alpha]]_k^{[m,n]} & := & \bigvee_{i=1}^{N_k(\varphi)}((w_{m,n} \equiv w_{0,i}) \wedge \\
 & & (\bigvee_{l=0}^{k} L_{k,i}(l) \vee dead(w_{k,i})) \wedge \\
 & & \bigwedge_{j=0}^{k'}(I_\varphi^j \rightarrow [[\alpha]]_k^{[j,i]})), \\
[[E(\alpha U_\varphi \beta)]]_k^{[m,n]} & := & \bigvee_{i=1}^{N_k(\varphi)}((w_{m,n} \equiv w_{0,i}) \wedge \\
 & & \bigvee_{j=0}^{k}((I_\varphi^j \wedge [[\beta]]_k^{[j,i]}) \wedge \\
 & & \bigwedge_{t=0}^{j-1}[[\alpha]]_k^{[t,i]})).
\end{array}$$

Let $[[\psi]]_{M_k}$ denote $[[\psi]]_k^{[0,1]}$ and use the arithmetic expressions to represent I_φ^j directly. $L_{k,i}(l) := T(w_{k,i}, w_{l,i})$ encodes a backloop from the kth state to the lth state in the ith k-path. k' in case EG is the same in Definition 6.

Take $EF_{=60} safe$ in bridge-crossing problem as an example. The encoding of this formula at $k = 2$ is as follows:

$$\begin{array}{ll}
 & [[EF_{=60} safe]]_2^{[0,1]} \\
= & \bigvee_{i=1}^{1}((w_{0,1} \equiv w_{0,i}) \wedge \bigvee_{j=0}^{2}(I_{=60}^j \wedge [[safe]]_2^{[j,i]})) \\
= & (w_{0,1} \equiv w_{0,1}) \wedge ((I_{=60}^0 \wedge [[safe]]_2^{[0,1]}) \vee \\
 & (I_{=60}^1 \wedge [[safe]]_2^{[1,1]}) \vee (I_{=60}^2 \wedge [[safe]]_2^{[2,1]})) \\
= & (w_{0,1} \equiv w_{0,1}) \wedge ((D_\pi[0..0] = 60 \wedge safe(w_{0,1})) \vee \\
 & (D_\pi[0..1] = 60 \wedge safe(w_{1,1}) \vee \\
 & (D_\pi[0..2] = 60 \wedge safe(w_{2,1}))
\end{array}$$

So, from the encodings of $[[M, \psi]]_k$, we have the following theorem.

Theorem 3 *Let M be a model, M_k a k-model of M and ψ a TECTL formula. Then, $M \models_k \psi$ iff $[[M^{\psi,s_0}]]_k \wedge [[\psi]]_{M_k}$ is satisfiable.*

4 Experiments

The experiments are done under RedHat Enterprise Linux AS release 4 (Nahant Update 4) on a 2-processor Xeon 3.0GHz machine with 32GB RAM. We use the SMT solver Yices 1.0.3[9]. We also use SAL 3.0[13], UPPAAL 4.0.6[16], TSMV 1.0[18], NuSMV 2.4.3[7] as comparable methods.

4.1 Fischer's Protocol

The system consists of n ($n \geq 2$) processes: Process 1 , ..., Process n, which compete for an access to the critical region. Each process in Fischer's protocol has one local clock x and can access the global pointer $lock$ ($lock := p$ to assign its process ID to the pointer). Initially, all processes are in states idle and $lock := nil$. Process i's transitions can be described as follows:

$$\begin{array}{l}
i = idle_i \wedge lock = nil \rightarrow (i, x_i) := (ready_i, 0); \\
i = ready_i \wedge x_i < B \rightarrow (i, lock, x_i) := (wait_i, p, 0); \\
i = wait_i \wedge lock \neq i \wedge x_i > A \rightarrow (i) := (idle_i); \\
i = wait_i \wedge lock = i \wedge x_i > A \rightarrow (i) := (critical_i); \\
i = critical_i \rightarrow (i, lock) := (idle_i, nil).
\end{array}$$

- i is a variable expresses the state which the ith process is in.
- $idle_i, ready_i, wait_i, critical_i$ represent the ith process's state $idle, ready, wait, critical$ respectively.
- x_i is the ith process's clock variable.
- $lock$ is the global pointer of the system.
- A, B are two integer variables. If $A \geq B$ the mutual exclusion property is true and if $A < B$ the property is false.

4.1.1 Experimental Results and Analysis

For n processes, we verify the negation of mutual exclusion property $\psi_1 : EF(\bigwedge_{i=1}^{n} critical_i)$. We use TA as the model and compare our method with SAL and UPPAAL. The experimental results with $A = 1$ and $B = 4000$ (ψ_1 is true) are shown in Table 1.

The 1st column shows the number of processes; the 2nd to 4th columns give the bound, time and memory used by SMT-based BMC method; the 5th and 6th columns display

	SMT-based BMC			SAL		UPPAAL	
n	k	Time	Size	Time	Size	Time	Size
2	6	0.024	4.47	0.160	8.84	0.440	1.03
3	8	0.077	4.99	1.021	11.93	0.446	1.54
4	10	0.192	5.79	2.373	15.07	0.766	2.05
5	12	0.421	6.77	6.241	18.25	0.868	3.08
6	14	1.055	8.27	11.612	21.47	1.056	4.10
7	16	2.495	10.46	34.234	24.73	2.110	8.21
8	18	4.585	12.32	64.865	28.03	14.674	29.75

Table 1. Experimental results for ψ_1.

the time and memory used by SAL; the 7th and 8th columns give the time and memory used by UPPAAL.

From Table 1 we can see that, SMT-based BMC method spend less time and memory than SAL and the time increasing trend of SMT-based BMC is much slower than SAL and UPPAAL with the increases of processes.

4.2 Bridge-crossing problem

The bridge-crossing problem is a famous mathematical puzzle with time critical aspects [21]. A group of four persons, called P1, P2, P3 and P4, have to cross a bridge at night. It is dark and they can only cross the bridge if they carry a lamp. Only one lamp is available and at most two persons can cross at the same time. Therefore any solution requires that, after the first two persons cross the bridge, one of them returns, bringing back the lamp with him for the remaining people. The four persons have different maximal speeds: here P1 crosses in 5 time units (t.u.), P2 in 10 t.u., P3 in 20 t.u. and P4 in 25 t.u. When a pair crosses the bridge, they move at the speed of the slowest person in the pair. Now, how much time is required before the whole group is on the other side (in the $safe$ state). The details of this problem can be found in [18]. We use TKS as the model in this problem.

4.2.1 Experimental Results and Analysis

We have tested two properties:

- ψ_2: $EF_{=60}(safe)$
  The formula expresses the property that the whole group will be safe in 60 t.u.. The property is true as the minimum time to cross the bridge is 60 t.u..
- ψ_3: $EF_{\leq 59}(safe)$
  The formula expresses the property that the whole group will be safe with less than 60 t.u.. It is obviously false.

The results are shown in Table 2 and Table 3. The 1st column shows the magnifications (e.g. "$\times 10$" means multiply every person's crossing time and the timing constraint in the property by 10.); the 2nd to 4th columns give the bound, time and memory used by SMT-based BMC; the 5rd and 6th columns display the time and memory used by TSMV ; the 7th and 8th columns give the time and memory consumed by NuSMV.

	SMT-based BMC			TSMV		NuSMV	
$\times T$	k	Time	Size	Time	Size	Time	Size
$\times 1$	5	0.023	4.34	0.006	1.82	0.084	6.17
$\times 10$	5	0.026	4.34	0.023	1.93	2.618	41.94
$\times 20$	5	0.024	4.34	0.044	2.06	8.945	45.48
$\times 50$	5	0.024	4.34	0.162	2.34	58.351	58.54
$\times 100$	5	0.022	4.34	0.466	2.76	236.662	110.71
$\times 200$	5	0.026	4.34	1.274	3.60	2608.600	204.68

Table 2. Experimental results for ψ_2.

	SMT-based BMC			TSMV		NuSMV	
$\times T$	k	Time	Size	Time	Size	Time	Size
$\times 1$	5	0.023	4.34	0.014	1.82	0.165	8.70
$\times 10$	5	0.026	4.34	0.082	2.06	5.305	42.04
$\times 20$	5	0.024	4.34	0.191	2.32	18.647	45.48
$\times 50$	5	0.024	4.34	0.643	3.03	121.671	60.62
$\times 100$	5	0.022	4.34	1.829	4.22	502.045	120.18
$\times 200$	5	0.026	4.34	5.918	6.64	6494.262	205.19

Table 3. Experimental results for ψ_3.

For ψ_2, the bound is 5, which means after 5 steps the system is in the *safe* state and the property is true. For ψ_3, the bound is also 5, that means after 5 steps, the system will never reach the *safe* state within 59 t.u.. This is because the minimal time that the system first in $safe$ state is 60 t.u. at $k = 5$, that means when $k < 5$ the system can not reach state $safe$ and when $k \geq 5$ the system reaches $safe$ state, as the duration of the system is monotonically increased, $D_\pi[0..k] \geq 60$, the timing constraint can not be satisfied any more. So $k = 5$ is enough for ψ_3.

From Table 2 and Table 3, we can see that SMT-based BMC is much faster than TSMV and NuSMV and the time SMT-based BMC used is almost the same with different magnifications, but TSMV and NuSMV are increased exponentially.

5 Related work and Conclusions

We have considered BMC for real-time systems based on SMT solver Yices to verification TECTL properties. Our SMT-based BMC can deal with two kinds of models: TA and TKS. We have compared it with some other real-time system model checkers. The experimental results indicate that the efficiency of our method is better than others, as there is no need to encode the clock variables into boolean formulae and can deal with the clock constraints more efficient.

For future work, some reduction methods [12, 25, 22] can be incorporated into our method to simplify the k-model and reduce the number of verification paths. Another direction is to work on efficient techniques for the verification of valid TECTL properties under the timed automaton model without reaching a high completeness threshold similar to that considered for LTL properties in [24].

References

[1] A. Armando, J. Mantovani and L. Platania. Bounded Model Checking of Software using SMT Solvers instead of SAT Solvers. SPIN 2006: 146-162.

[2] S. Aggarwal and R. Kurshan. Modeling Elapsed Time in Protocol Specification. PSTV 1983: 51-62.

[3] R. Alur, C. Courcoubetis and D. L. Dill. Model-Checking in Dense Real-time. Information and Computation 104(1): 2-34, 1993.

[4] A. Biere, A. Cimatti, E. Clarke, M. Fujita, Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. Proceedings of ACM/IEEE Design Automation Conference (DAC99): 317-320.

[5] A. Biere, A. Cimatti, E. Clarke, Y. Zhu. Symbolic Model Checking without BDDs. TACAS99: 193-207.

[6] R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, C-35(12): 1035-1044, 1986.

[7] A. Cimatti, E. M. Clarke, F. Giunchiglia and M. Roveri. NuSMV: A New Symbolic Model Verifier. CVA 1999: 495-499.

[8] D. L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. Automatic Verification Methods for Finite State Systems 1989: 197-212.

[9] B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). CAV 2006: 81-94.

[10] E. A. Emerson and E. M. Clarke. Using Branching-time Temporal Logics to Synthesize Synchronization Skeletons. Science of Computer Programming 2(3): 241-266, 1982.

[11] E. A. Emerson, A. Mok, A. P. Sistla and J. Srinivasan. Quantitative Temporal Reasoning. CAV 1990: 136-145.

[12] E. A. Emerson and A. P. Sistla. Symmetry and model checking. Formal Methods in System Design, 9: 105-131, 1995.

[13] http://sal.csl.sri.com/index.shtml.

[14] F. Jahanian and A.Mok. Safety Analysis of Timing Properties in Real-time Systems. IEEE Transactions on Software Engineering 12(9): 890-904, 1986.

[15] R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. Real-Time Systems 2(4): 255-299 (1990).

[16] K. G. Larsen, P. Pettersson, and Wang Yi. UPPAAL in a nutshell. J. Software Tools for Technology Transfer, 1(1-2): 134-152, 1997.

[17] N. A. Lynch and H. Attiya. Using Mappings to Prove Timing Properties. PODC 1990: 265-280.

[18] N. Markey and P. Schnoebelen. Symbolic Model Checking for Simply-timed Systems. FTRTFT 2004: 102-117.

[19] W. Penczek, B. Wozna, and A. Zbrzezny. Bounded Model Checking for the Universal Fragment of CTL. Fundamenta Informaticae 51: 135-156, 2002.

[20] W. Penczek, B. Wozna and A. Zbrzezny. Towards Bounded Model Checking for the Universal Fragment of TCTL. FTRTFT 2002: 265-290.

[21] G$\ddot{u}$nter Rote. Crossing the bridge at night. Bulletin of the EATCS, 78: 241-246, 2002

[22] L. Xu, W. Chen, Y. Xu, W. Zhang. Improved Bounded Model Checking for the Universal Fragment of CTL. JCST. To appear.

[23] F. Yu, B. Y. Wang and Y. W. Huang. Bounded Model Checking for Region Automata. FTRTFT 2004: 246-262.

[24] W. Zhang. SAT-Based Verification of LTL Formulas, FMICS 2006.

[25] C. Zhou, Decheng Ding. Improved SAT Based Bounded Model Checking. TAMC 2006: 611-620.